_ TELECOM ESCUELA
TÉCNICA VLC SUPERIOR
DE UPV INGENIEROS
DE TELECOMUNICACIÓN

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

TRABAJO FIN DE GRADO

Redes Definidas por Software (SDN): OpenFlow

Autor: David Andrés Serrano Carrera

Tutor: Juan Carlos Guerri Cebollada

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS
DE TELECOMUNICACIÓN

Valencia, 2015

Resumen

El presente trabajo pretende dar una visión amplia de las "Redes Definidas por Software (SDN)", concepto innovador en las arquitecturas de datos. Se describe la evolución y desencadenantes de la implementación más aceptada, OpenFlow.
Además, a modo educativo y visual, se trabaja con Mininet, un software de emulación de redes virtuales pensado para experimentar y comprobar los aspectos teóricos de SDN y OpenFlow. En concreto, se realizan pruebas para determinar la importancia de un elemento clave como es el controlador en ésta nueva arquitectura; se trabaja con MiniEdit, una interfaz gráfica que hace uso del potencial de Mininet y finalmente se construye de forma básica un controlador POX que realiza las funciones de un conmutador de capa 2.

Resum

El present treball pretén donar una visió àmplia de les "Xarxes Definides per Software (SDN)", concepte innovador a les arquitectures de dades. Es descriuen l'evolució i els desencadenants de la implementació més acceptada, OpenFlow.
A més, de forma educativa i visual, es treballa amb Mininet, un software d'emulació de xarxes virtuals pensat per a experimentar i comprovar els aspectes teòrics de SDN i OpenFlow. En concret, es realitzen proves per a determinar la importància d'un element clau com ho és el controlador en aquesta nova arquitectura; es treballa amb MiniEdit, una interfície gràfica que fa ús del potencial de Mininet i finalment es construïx de forma bàsica un controlador POX que realitza les funcions d'un commutador de capa 2.

Abstract

This report aims to give a broad view of the innovative concept in data architectures "Software Defined Networks (SDN)". Evolution and triggers about the most accepted implementation, OpenFlow, are described.
In addition, in a education and visual way, it works with Mininet, a emulation software of virtual networks designed to experiment and verify the theoretical aspects of SDN and OpenFlow. Specifically, tests are done to determine the importance of a key element, the controller, on this new architecture; MiniEdit, a graphical interface that makes use of the Mininet potential, and finally it builds in a basic way a POX controller that performs the functions of a layer 2 switch.

Índice

1. Introducción

Para entender la aparición de SDN y OpenFlow, es preciso explicar la tecnología relacionada con las redes, así como la necesidad de evolucionar ante la aparición de nuevas exigencias.

1.1 Programación en la red: Redes Activas

La red más importante de todas, Internet, comenzó como un proyecto creado a principios de los años 70 por la Agencia para Proyectos Avanzados del Departamento de Defensa Norteamericano (DARPA) con el objetivo de conectar los equipos de las instalaciones del gobierno de los Estados Unidos. En ésta época se llamaba ARPANET, y cuando universidades y centros de investigación se unieron, la red pasó a tener un carácter más de investigación y desarrollo.

En la década de los 90, Internet pasaba de ser una red casi exclusivamente académica a abrirse al gran público, motivada por diversos factores [1]:

- Aparición de ordenadores personales, por lo que empresas y hogares empezaron a tener equipos a precios más asequibles.

- Mejora de las redes de telecomunicaciones, dando como resultado velocidad y calidad aceptables, empezando por áreas urbanas.

- Se establecieron y estandarizaron prácticas y servicios en la red (aparición de organizaciones como el Grupo de Trabajo de Ingeniería de Internet, IETF).

- Nuevos servicios multimedia e interfaces más accesibles e intuitivas (como la *World Wide Web*, WWW).

- La enorme cantidad de información disponible, haciendo a la red muy atractiva.

Como resultado, los investigadores querían probar e implementar nuevas características en la red. Sin embargo existían algunos obstáculos, como el lento proceso de estandarización que obligaba a los investigadores a probar las novedades en pequeños laboratorios para después, si el financiamiento y la

motivación continuaba, llevar estas ideas al IETF [2]. El ciclo de maduración e implantación de protocolos de red suele ser lento (de cinco a diez años) y el principal ejemplo actual es IPv6. Además, tiene gran importancia el nivel de acogida de las novedades que los fabricantes quieran dar a sus equipos de red [3].
Se desarrollaron entonces diferentes soluciones alternativas, siendo el proyecto de Redes Activas a mediados de los años 90 el más importante en el intento de despliegue de nuevos servicios de red. A pesar de que había trabajo previo como SoftNet [18], una red experimental multisalto que introdujo la idea de añadir comandos al contenido del paquete, éste era muy disperso y se necesitaban ciertos aspectos comunes, como los términos utilizados o la definición de un nodo de red genérico.
Además, esta tecnología se vio fuertemente impulsada por los avances en lenguajes de programación como Java, la tecnología de máquina virtual, o programas anteriores en búsqueda del "Internet del Futuro" como GENI (*Global Environment for Network Innovation*), una red programable en Estados Unidos aún vigente y diseñada para pruebas, FIRE (*Future Internet Research and Experimentation*) o FIND (*Future Internet Design*) [4].

Impulsado en un programa también de DARPA ("*DARPA Architectural Framework for Active Networks*"), la idea básica de las Redes Activas es permitir la programabilidad de la red a través de una interfaz programable, ya que las redes convencionales de paquetes son insensibles a los datos que transportan: se procesa la cabecera y se encamina en base a ella. Terceros como usuarios finales, operadores y proveedores de servicio pueden proporcionar o adaptar servicios específicos a cada aplicación en forma de código que informa de la red. El código es aceptado solo si no compromete ninguna característica como el rendimiento o la seguridad [8] (almacenamiento de un programa en la memoria temporal del nodo, bucles infinitos, acceso a zonas del nodo restringidas, etc.).

Como se ha dicho, la programabilidad es el objetivo fundamental, haciendo que se controle dinámicamente la red mediante mecanismos de distribución y ejecución del código en lugar de una manera determinada como los algoritmos de enrutamiento. Es importante destacar que al tratarse de un modelo experimental, los nodos de la red deben poder tratar también paquetes tradicionales a una velocidad similar a la de los equipos de red típicos.
La distribución de código en un nodo se puede realizar de dos formas:

- Modelo discreto, de cápsula o "fuera de banda". Se separa el paquete de datos del programa en sí (extensiones activas), analizando su cabecera y ejecutándolo para saber cómo tratar a dicho paquete. Este modelo es

menos “activo”, ya que las decisiones las toman los administradores de red y no usuarios finales, haciéndolo también más fácil de implementar en las redes actuales.

- Modelo integrado, de enrutador programable o “en banda”. Los paquetes son activos en conjunto, incluyendo datos y programa (código embebido, en caso de tener un tamaño pequeño o ser paquetes poco frecuentes y carga bajo demanda, en el que un proceso es cargado solo la primera vez que se recibe un paquete de ese tipo específico).

Los servicios usan la interfaz programable del nodo, la implementan como una API (*Application Programming Interface*) de red para gestionar dinámicamente sus recursos y definen una máquina virtual que interpreta un lenguaje específico [5], el cual determina a la API y cómo programar la red, ya que de su elección se puede “prever” el comportamiento que tendrá un determinado programa en el nodo.
Se suele citar una serie de propiedades que debería poseer un lenguaje de programación válido para Redes Activas [6], siendo las más destacables: fuerte tipado (no permitir la violación de los tipos de datos, prohibir el uso de una variable de un tipo en otra a menos que se realice una conversión), tener un recolector de basura, cargar dinámicamente código, tener un mecanismo que permita definir diferentes interfaces para acceder a un mismo módulo, ofrecer buen rendimiento y permitir la creación de servicios mediante la composición de bloques básicos o “componentes” [5].

La arquitectura propuesta por el programa DARPA tiene como objetivo definir los principales componentes e interfaces en una red activa y que se pueden ver en la figura 1, así como la posibilidad de varias APIs de red debido a que no hay una solución única o que pueda incorporar todas las funcionalidades deseadas. En un nodo, el usuario demanda un servicio que se traduce en una Aplicación Activa (AA), la cual provee el código necesario para programar un Entorno de Ejecución (EE). Este EE junto con el Sistema Operativo de Nodo (NodeOS) sirven de “intermediarios”, ya que acceden, gestionan y comparten los recursos físicos (transmisión, procesamiento, almacenamiento) y ocultan la interacción con el usuario. Cada nodo activo posee también un EE específico de gestión para controlar aspectos de su configuración como bases de datos con políticas de seguridad o permitir el arranque de otros EEs y por tanto, de otras aplicaciones. Para que el usuario especifique a qué EE va dirigido cada paquete se usa el *Active Network Encapsulation Protocol* (ANEP) mediante un campo en su cabecera.

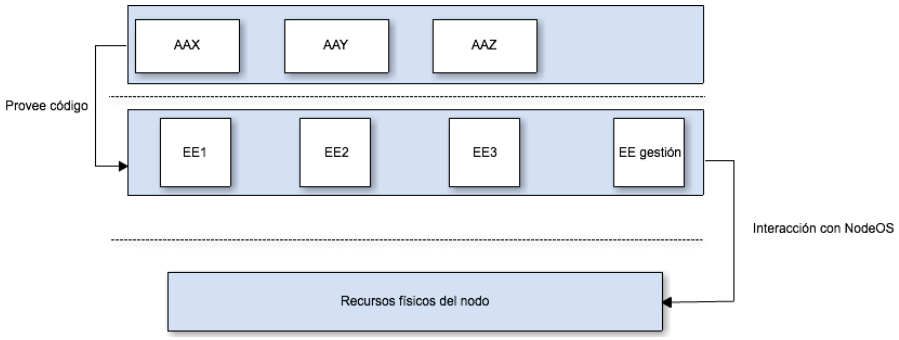

Figura 1. Componentes del nodo activo [5][7]

Destacan diversos ejemplos concretos de redes activas:

- Switchware, de la Universidad de Pensilvania. Se permite cualquiera de los dos modelos de distribución de código anteriormente citados, tanto código embebido en lenguaje PLAN (*Programming Language for Active Networks*), como extensiones activas (*switchlets*).

- ANTS, del MIT. Utiliza el modelo de descarga bajo demanda. La API es una máquina virtual Java (JVM) extendida con algunas clases propias para el tratamiento con paquetes activos y para la seguridad. Además, los nodos activos pueden disponer de memorias *soft-state* (contienen información temporal).

- SmartPackets, de BBN. Se usa el modelo de código embebido. La API es una máquina virtual Spanner (lenguaje ensamblador) y utiliza otro lenguaje similar a C, Sprocket, que mejora la seguridad de la red.

La ya comentada programabilidad de la red activa permite que los servicios demandados por el usuario se puedan adaptar dinámicamente a cambios, mejorando el rendimiento con respecto a soluciones tradicionales extremo a extremo.

A pesar de las grandes expectativas, las Redes Activas no contaron con un gran despliegue debido a factores como: la sensación de estar ante un modelo más teórico que práctico, el escepticismo de los fabricantes a que usuario finales controlaran la red o el desarrollo del plano de datos (API de red) dejando de lado el plano de control en el nodo. Con todo, este primer acercamiento a una visión de la red como un conjunto de nodos personalizables es básica en SDN, ayudó a que los investigadores vieran que se podía innovar y sentó las bases para poder aislar el tráfico experimental del tradicional.

1.2 La importancia de evolucionar en los planos de control y de datos

A principios de los 2000, las redes seguían creciendo a gran ritmo debido al avance en su fiabilidad, velocidad y rendimiento, con la aparición de tecnologías como ADSL que proporcionaban acceso a Internet para hacer posible el uso de diferentes servicios: correo electrónico, servicios de teleconferencia, o intercambio de archivos multimedia de mayor tamaño entre los usuarios. La programabilidad intentada en años anteriores se vio potenciada también por la concepción y gran desarrollo en un elemento de hardware básico como es el procesador de red [14], un circuito integrado programable por software diseñado para aplicaciones de red como el tratamiento de paquetes, funciones de control de acceso y seguridad o calidad de servicio identificando diferentes tipos o clases de paquetes.

Es importante citar también el avance en la gestión y almacenamiento de la información, apareciendo nuevos conceptos como el *cloud computing* [19], paradigma de dar servicios a través de Internet, almacenando la información en servidores en la red que aceptan peticiones de usuarios que tienen "copias" de los recursos; la mejora de la velocidad de los servidores en comparación con los dispositivos de enrutamiento; o el mayor tamaño de los centros de procesamiento de datos o *data centers*, ubicaciones donde se concentran los recursos necesarios para procesar la información.
Así, mientras las redes seguían creciendo en tamaño o velocidad, el proceso de enrutamiento seguía teniendo problemas importantes como el control del comportamiento de los dispositivos o la heterogeneidad en los procesadores de red, haciendo necesario algún tipo de acuerdo a la hora de implementar el plano de datos o reenvío y su interacción con el plano de control. Los investigadores veían necesaria la innovación en este sentido, y centraron sus esfuerzos en intentar evitar la alta integración que había entre ambos planos en el enrutamiento.

Un enrutador tradicional realiza fundamentalmente dos tareas:

- Conocer y actualizar la topología de la red en cada momento, y almacenar esta información en tablas de forma local. Éstas se actualizan mediante, por ejemplo, protocolos de enrutamiento, que eligen la mejor ruta o "path" a seguir por los paquetes. Dicha tarea se realiza en el plano de control.

- Reenviar los paquetes entrantes según las decisiones en el plano de control. Esto se realiza en el plano de datos o de reenvío.

En la figura 2 se puede ver el proceso de un paquete entrante en un enrutador, según si tiene que tratarlo, si lo ha generado localmente el dispositivo, o si solo tiene que realizar la tarea de reenviar, como en un conmutador o "switch".

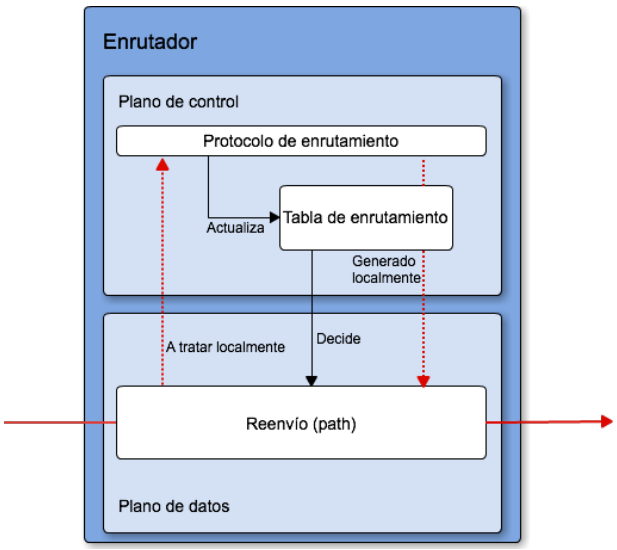

Figura 2. Procesado de un paquete entrante en un enrutador tradicional [9]

En el ámbito de la comunicación entre los planos de control y datos también había trabajo previo, como la iniciativa estándar IEEE P1520 [12] para Interfaces de Redes Programables, que identifica la necesidad de una interfaz programable para la red y la forma de acceder a la misma. Existían proyectos simultáneos en la época, como la arquitectura SoftRouter [10], que definiría la vinculación dinámica entre elementos de red y que compartió el concepto de una interfaz estándar entre el plano de control y de datos con la solución del grupo de trabajo del IETF, *Forwarding and Control Element Separation* (ForCES).
ForCES [11] presenta una estructura lógica basada en dos entidades básicas como se puede apreciar en la figura 3:

- Elemento de reenvío (FE). Usa el hardware para procesar y manejar los paquetes, dirigido por un CE.

- Elemento de control (CE). Indica a uno o más FEs cómo procesar los paquetes. Además ejecuta protocolos de señalización y control.

Estos elementos se ven complementados por dos entidades de gestión, Gestor FE (opera durante un periodo en el que se determina con qué CE se debe comunicar un FE, llamado fase de pre-asociación) y Gestor CE (opera también durante la pre-asociación y determina con qué FEs debe comunicarse un CE).

En la fase de post-asociación el control del CE sobre un FE ya está determinado.

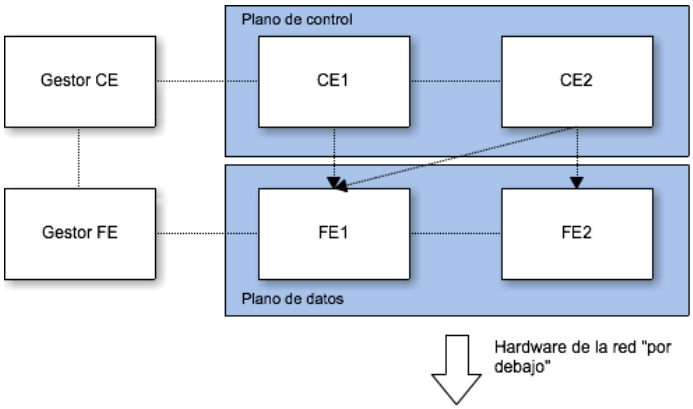

Figura 3. Arquitectura ejemplo en ForCES [11]

ForCES no fue adoptado por muchos de los fabricantes de enrutadores, sobre todo porque continuaba el problema de no encontrar una API estándar; el recelo ante el cambio de tener una visión común a una local en el dispositivo (comprometiendo el *fate sharing*, dependencia entre todas las partes de la red); o la necesidad de un protocolo de enrutamiento existente.
Se continuaron ideando modelos nuevos como el proyecto 4D [2], que se basaba en cuatro planos; plano de decisión, responsable de crear una configuración de red; plano de diseminación, que entrega información sobre la visión de la red al plano anterior; plano de descubrimiento, que permite a los dispositivos de red descubrir a sus vecinos; y un plano de datos responsable de procesar paquetes.
Esta arquitectura tuvo su implementación en Tesseract [13], permitiendo el control directo a través de un elemento de decisión que creaba explícitamente el estado de reenvío a los nodos de la red bajo un solo dominio administrativo, en lugar de políticas de enrutamiento indirectas como por ejemplo, los pesos de los enlaces en OSPF.

También cabe destacar el último gran proyecto anterior a SDN, que es SANE/Ethane [15][16]. En 2007, investigadores de las Universidades de Stanford y Berkeley idearon una arquitectura de red para empresas que adoptó las ideas expresadas en 4D de un control centralizado, añadiendo mecanismos de seguridad. Una característica muy importante era que Ethane permitía su despliegue junto con dispositivos tradicionales, haciendo que la adopción de esta tecnología fuera fácil. Se concibieron conmutadores Ethernet basados en flujos, los cuales eran gestionados gracias a un control centralizado que se comunica con los dispositivos a través de un canal seguro. Por este concepto de

conmutador “simple” con flujos, Ethane es considerado el predecesor de OpenFlow y de su API inicial.

1.3 Situación actual

A finales de los 2000 y ya en la actualidad, la complejidad de las redes sigue aumentando; son más grandes, y se necesita un despliegue de servicios más dinámico, ya que las redes tradicionales no permiten programabilidad, flexibilidad o capacidad de probar nuevas ideas sin interrumpir los servicios que se estén ejecutando. Se desarrollan y mejoran mecanismos de control como listas de control de acceso (ACLs), cortafuegos, ingeniería de tráfico (distribuir sobre toda la topología de red el tráfico para evitar congestión) o virtualización de las redes (VLANs), entre otros.

Concebida para planificar, organizar y controlar los dispositivos con el fin de garantizar un servicio de acuerdo a un coste, la gestión de la red es una tarea tediosa y compleja para cualquier administrador, y su objetivo es que la red esté disponible el máximo tiempo posible mediante la corrección y detección de fallos o monitorización de los dispositivos y recursos de la red como servidores, estaciones de trabajo equipos de interconexión tales como enrutadores o conmutadores, entre otros.
Las primeras redes tenían una gestión autónoma, es decir, cada nodo se gestionaba localmente, teniendo un coste sobre todo por las comunicaciones entre ellos y hacia el administrador, al tener que tomar decisiones que afectaban a otros nodos. Con el paso del tiempo, los fabricantes de equipos entendían que se debía aplicar una gestión homogénea con un único nodo de gestión centralizado. Sin embargo, con el crecimiento de las redes y la aparición de nuevos fabricantes, las redes no tenían equipos y protocolos homogéneos no solo entre distintos vendedores, sino en dispositivos de una misma marca, por lo que se pasó a una gestión heterogénea, con los problemas de eficiencia y de coste que conlleva: datos duplicados, múltiples interfaces o baja innovación por la complejidad de la red.

Los servidores y centros de datos dedicados están siendo sustituidos por *cloud datacenters*, los recursos están en la nube (con problemas de acceso eficiente) y representan un gran avance en cuanto a coste de mantenimiento. Los servidores son desplegados en minutos o segundos; pueden ser movidos de un host físico a otro con un solo clic usando tecnologías como vMotion [20], mientras que el

almacenamiento emplea una estructura más lógica en lugar de física, aumentando la flexibilidad.
Sin embargo, las tareas de configuración y mantenimiento de las redes han seguido siendo complejas: un administrador de red accede mediante la interfaz de línea de comandos (CLI) a los dispositivos, con el consecuente consumo de tiempo, tendencia al error (es el responsable de alrededor del 70% del fallo en los dispositivos de red [21]) y las dificultades en el caso de trabajar con dispositivos de distintos fabricantes, ya que al añadir uno nuevo, se necesita una configuración coherente con la red entera.

Surgen también nuevas exigencias por parte de los usuarios, como una gran movilidad, seguridad, el "estar siempre conectado" o aplicaciones 'Big Data', término que hace referencia al manejo de datos tan grandes o complejos que las redes tradicionales ya no son suficientes para su uso. Ya no solo se trata del volumen, sino también de la gran variedad de información disponible en dispositivos móviles, GPS, contenidos multimedia de alta definición (streaming de vídeo a definición ultra-alta, UHD ó 4K), sensores eléctricos, medidores, etc. que además deben comunicarse a un gran velocidad.
En 2014, el tráfico de datos móviles en el mundo creció un 69% respecto al año anterior, llegando a 2.5 exabytes por mes. En el mismo año, el número de dispositivos móviles conectados excedió el de personas en el mundo. Las velocidades de las conexiones móviles de red se duplicarán para 2019 (1.7 Mbps en 2014 a cerca de 2 Mbps en 2019) y más de la mitad de todos los dispositivos conectados a la red móvil serán dispositivos inteligentes. Además, 3/4 del tráfico de datos móviles mundial será vídeo en 2019 [17][22].
A continuación se muestran datos y estadísticas que confirman esta evolución.

Año	Tráfico global de Internet
1992	100 GB por día
1997	100 GB por hora
2002	100 GB por segundo
2007	2000 GB por segundo
2014	16144 GB por segundo
2019	51794 GB por segundo

Tabla 1. Contexto histórico del tráfico en Internet [22]

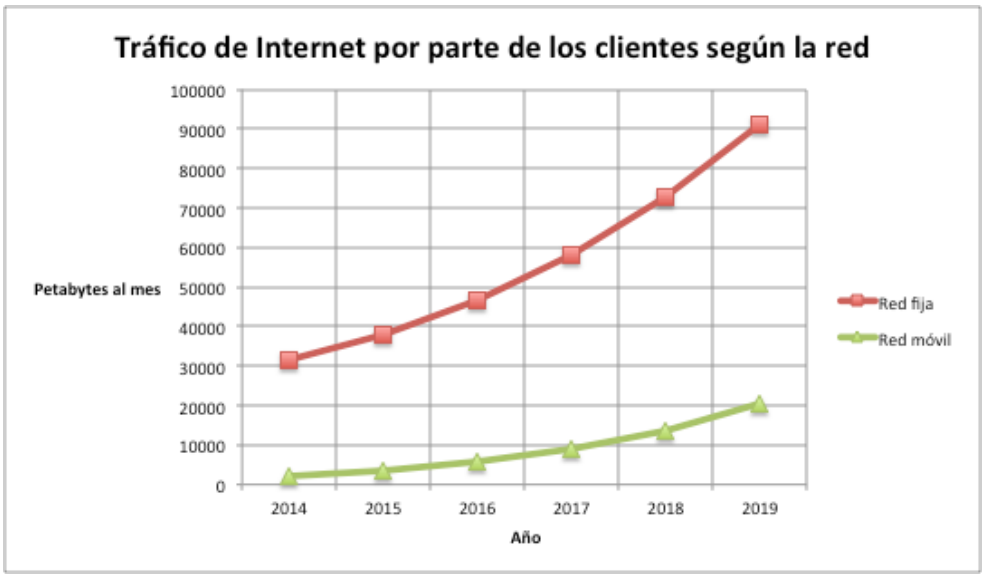

Figura 4. Tráfico de Internet según el tipo de red [22]

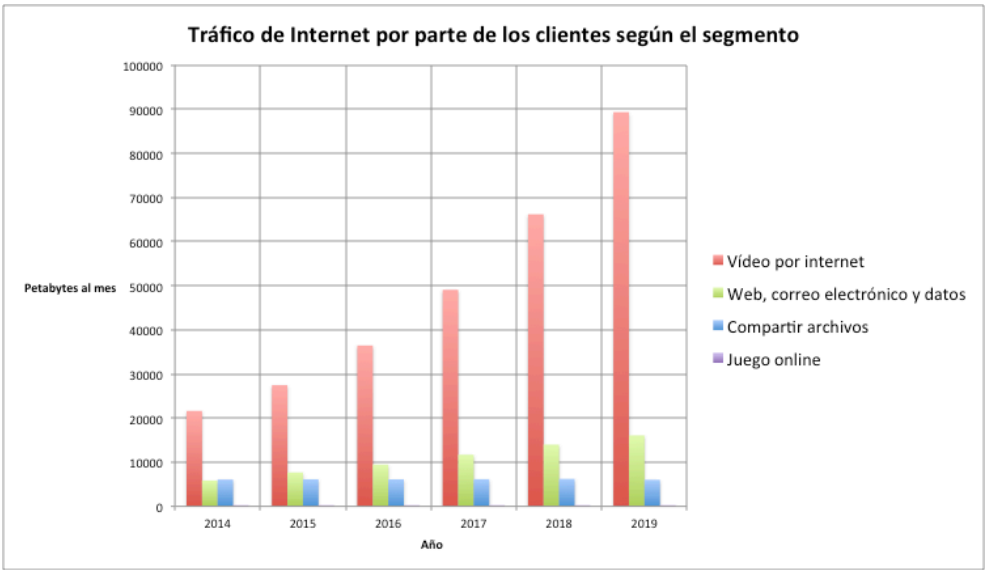

Figura 5. Tráfico de Internet según el segmento [22]

Por tanto, se puede comprobar la tendencia de los usuarios a utilizar redes inalámbricas y mayor tamaño del tráfico de datos, con los consecuentes problemas de escalabilidad ya mencionados.

2. Redes Definidas por Software, SDN

SDN surge de todo el trabajo anteriormente citado en relación a la programabilidad de la red y separación de los planos de control de datos, y es una propuesta para solucionar la evolución de los dispositivos de red, concibiendo una interfaz abierta entre los planos así como software de terceros, ya que la reticencia de los fabricantes y su poder en la evolución de las redes han sido un obstáculo constante, sobre todo al tener una industria verticalmente integrada, es decir, poco receptiva a la innovación y a la competencia en el mercado.

En 2011, Deutsche Telekom, Facebook, Google, Microsoft, Verizon y Yahoo! crearon la *Open Networking Foundation* (ONF) [23], una organización sin ánimo de lucro dedicada a la promoción y adopción de SDN a través del desarrollo de estándares abiertos y que se apoyaba en el trabajo previo de investigadores de las Universidades de Stanford y Berkeley. Como se detallará más adelante, propone el estándar OpenFlow como el despliegue comercial de este nuevo concepto.

La idea fundamental de las Redes Definidas por Software (SDN) es clara: trasladar el plano de control del plano de datos en los dispositivos físicos (enrutadores y conmutadores) unificándolo en un solo elemento externo a la red física, el controlador.
En lugar de usar protocolos distribuidos como OSPF o BGP, el controlador es el componente software común encargado de decidir el reenvío y construir las tablas de enrutamiento de los dispositivos de red (convirtiéndolos en más simples que los actuales), así como controlar la red desde un solo punto, lo que significa poder también configurar todo tipo de equipos (enrutadores, conmutadores, traductores de direcciones de red NATs o cortafuegos) mediante un software de gestión estándar, sin fabricante, haciendo que la red se transforme en un sistema abierto [24].
Además, se tiene una visión completa, ya que se intercambia información constantemente entre las diferentes capas de la arquitectura: aplicación, control e infraestructura, como se observa en la figura 6 y a continuación se detalla.

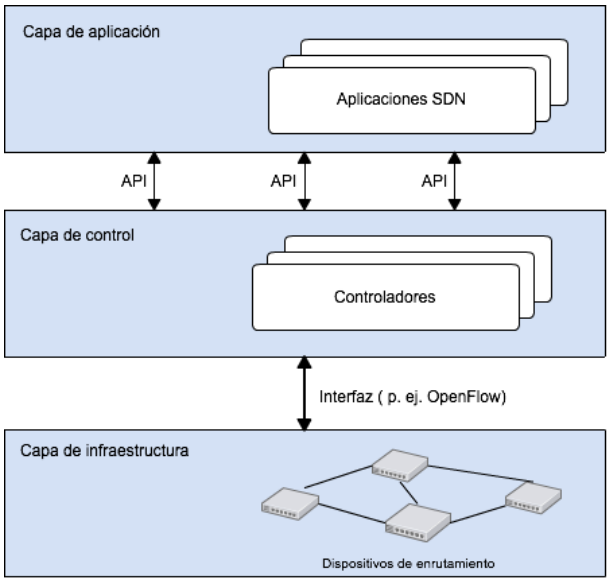

Figura 6. Modelo de la arquitectura SDN [25]

2.1 Capa de Aplicación

Las aplicaciones SDN comunican sus requisitos a la red a través de una API que conecta con la capa de control, y están diseñadas para satisfacer las necesidades de los usuarios.
Ejemplos de aplicaciones SDN:

- Enrutamiento adaptativo. Tradicionalmente el enrutamiento conlleva una compleja implementación y convergencia lenta. Con SDN han surgido dos conceptos populares: balance de carga, a través de diversas propuestas de algoritmos que solucionen el alto coste de balanceadores de carga dedicados, y el diseño de intercambio de información entre capas, para mejorar la integración entre entidades y proporcionar calidad de servicio (QoS), con *Qos-aware Network Operating System* (QNOX) [26] como principal ejemplo.

- Itinerancia sin interrupciones. La transferencia o *handover* al hacer uso de dispositivos móviles hace necesario prever un servicio continuo. Con SDN, las redes entre diferentes portadores con diferentes tecnologías tienen un plano de control común. Hay diferentes propuestas de transferencia como Hoolock en redes Wi-Fi y WiMAX u Odin para WLANs de empresa.

- Mantenimiento de la red. Herramientas de configuración típicas como traceroute o tcpdump no son una solución para mantener una red extensa de forma automática, ya que, como se ha citado, tienden al error humano. Al tener una visión global de la red y un control centralizado de la configuración, SDN permite nuevas herramientas de diagnóstico como ndb u OFRewind.

- Seguridad de la red. Las redes tradicionales utilizan cortafuegos o servidores proxy para proteger las redes físicas, siendo su implementación una tarea pesada para el administrador [25] . SDN permite analizar patrones de tráfico para posibles problemas de seguridad como ataques de denegación de servicio, guiar paquetes sospechosos a sistemas de prevención de intrusión (IPS), modificar reglas de reenvío para bloquear tráfico, o dar privacidad a los usuarios con ejemplos como AnonyFlow.

- Virtualización de la red. Lo que se pretende es permitir la existencia en una infraestructura compartida de múltiples arquitecturas de red heterogéneas. Normalmente, lo que se hace es separar la red física en múltiples instancias virtuales y asignarlas a diferentes usuarios, controladores o aplicaciones, mediante túneles o etiquetas VLAN y MPLS, convirtiéndose en una tarea compleja. Con SDN, la configuración se realiza en el controlador con plataformas como libNetVirt, una librería de virtualización de red o FlowVisor, colocando un proxy transparente para filtrar mensajes de control según la red virtual.

- Cloud Computing. Los centros de datos en las redes para 'cloud computing' necesitan algunas características, como escalabilidad, independencia de la localización para abastecer recursos dinámicos o diferenciación de QoS. La conmutación virtual es usada para la comunicación entre máquinas virtuales en el mismo host e implementada en SDN como Open vSwitch.

2.2 Capa de Control

Lógicamente centralizado, el controlador es el componente más importante de la arquitectura SDN ya que gestiona la capa de aplicación e infraestructura mediante dos interfaces, una con cada plano adjunto. Además, debe monitorizar todos los elementos para dar una visión global de la red.

Mediante la comunicación con el plano de infraestructura, se recoge el estado de la red y, según las exigencias de las aplicaciones, actualiza en los dispositivos las reglas de reenvío, ya que pueden producirse cambios como recuperación tras un fallo, migración de máquinas virtuales o balance de carga. También se debe mantener una validez y consistencia a fin de evitar bucles o agujeros de seguridad.
Por otra parte, se comunica con las aplicaciones SDN mediante la "traducción" de sus requisitos con un lenguaje de alto nivel, teniendo varias opciones como lenguajes existentes (Python, Java, C++), librerías en un kit software de desarrollador (SDK) como OnePK de Cisco, o lenguajes nuevos como *Flow-based Management Language* (FML), Frenetic (similar a SQL) o Nettle.
Además, el estado de la red (basado en número de paquetes, tamaño de los datos o ancho de banda en un flujo) que se recoge de la capa de infraestructura se debe comunicar a la aplicaciones para informarlas y construir, por ejemplo, un gráfico con la topología existente. En este sentido, la manera más común de hacerlo es mediante una Matriz de Tráfico (TM), que representa todos los flujos entre los posibles pares orígenes y destinos de la red.

En cuanto a la capa de control en sí, se necesita que un controlador sea capaz de comunicarse con otros debido a que:

- No se tiene un controlador estándar en SDN, y en una red grande pueden existir diversas opciones, por lo que es necesaria la transferencia de información para la visión global de la red y la toma de decisiones en la misma.

- Un solo controlador puede ocasionar problemas de congestión ya que se trata de un punto crítico. Se puede tener controladores de 'backup' o de réplica.

La solución más citada en este sentido es HyperFlow, que proporciona una vista sincronizada y consistente entre múltiples controladores.

Para finalizar, hay que destacar algunas características de los controladores:

- Pueden aparecer conflictos de configuración al haber una comunicación constante entre aplicaciones y controladores, por lo que hay diferentes soluciones propuestas como FlowChecker, NICE (basado en verificación de modelos, un método para sistemas formales) o VeriFlow.

- Al tratarse de software, se puede mejorar el rendimiento del controlador con técnicas de optimización como *batching* o procesamiento por lotes

(sin control de usuario). Diversos controladores como Maestro (basado en Java), NOX-MT (basado en la implementación C++ de NOX) o McNettle (en lenguaje Haskell) lo utilizan.

- Se puede comparar el rendimiento de controladores mediante herramientas como Cbench u OFCBenchmark.

2.3 Capa de Infraestructura

Esta capa consta de los dispositivos hardware de conmutación que forman una red y realizan dos tareas de acuerdo a sus dos componentes lógicos:

- Control. Recoge el estado de la red (topología o estadísticas de tráfico) y se lo comunica al controlador, el cual a su vez le indica las reglas de reenvío de paquetes. Esta información es almacenada temporalmente en la memoria local (como Memoria Direccionable de Contenido Ternario, TCAM o Memoria de Acceso Aleatorio Estático, SRAM), elemento fundamental de esta entidad lógica, ya que en caso de ser insuficiente los paquetes serían descartados o enviados directamente al controlador, con una obvia degradación de la red. Hay diferentes soluciones a este respecto como la agregación de las rutas (mediante un prefijo común) o algoritmos para optimizar la caché.

- Datos. El procesador de red reenvía los paquetes en base a las decisiones tomadas por el plano de control; al recibir un paquete, el dispositivo identifica la decisión de reenvío que coincide con el paquete. Además de realizar el proceso en base a las direcciones IP o MAC como en la redes tradicionales, en SDN puede ser también basándose en puertos TCP/UDP, etiqueta VLAN o puerto de entrada en el conmutador.

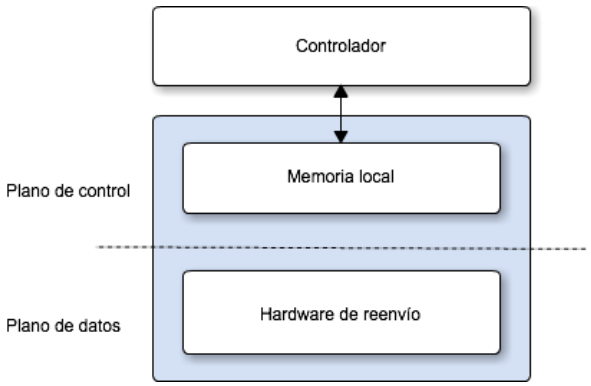

Figura 7. Dispositivo en la capa de infraestructura [25]

Es importante destacar el comportamiento de un dispositivo SDN al recibir un paquete: el primer paquete de un flujo se envía al controlador para su procesado e inspección de la cabecera del paquete, actualizando las tablas de reenvío. Una vez llegan más paquetes de dicho flujo, éstos se reenvían de acuerdo a la tabla, lo que se traduce en mayor sencillez al fabricarlos y menor coste.

Finalmente, cabe destacar ejemplos reales de implementaciones de dispositivos SDN según su naturaleza:

- Implementación en un PC. Se trata de software ejecutándose en un sistema operativo anfitrión como Linux. Existen ejemplos como Pantou, Open vSwitch u OpenFlowClick. La principal desventaja es la lenta velocidad de procesado de paquetes debido al número limitado de puertos a las tarjetas de red (NICs), aunque pueden proporcionar conmutación virtual a máquinas virtuales (VMs) para satisfacer las necesidad de virtualización del servidor o *cloud computing* [25].

- Implementación en hardware de red abierto. Independiente del fabricante y con una plataforma programable para construir redes para la investigación y experimentos. Existen ejemplos como SwitchBlade y ServerSwitch (basados en el proyecto abierto NetFPGA) u ORAN (basado en las especificaciones de Arquitectura Avanzada de Computación en Telecomunicaciones, ATCA). Es lo más común para experimentar en los laboratorios con SDN, ya que son más flexibles que los dispositivos de fabricantes y con más capacidad que los basados en software.

- Implementación por parte de un fabricante. Dispositivos como NEC PF5240, IBM G8264, Pica8 3920, o actualización de versiones anteriores como Indigo.

3. OpenFlow

En 2008, los investigadores de la Universidad de Stanford Nick McKeown et al. idearon OpenFlow, una nueva forma en la que experimentar en las redes actuales. A pesar de que en las primeras propuestas como [27] no se define la relación entre SDN y OpenFlow, la concepción de un software de control separado de la red fue visto como una implementación real de SDN, estandarizando la forma en la que el controlador se comunica con los dispositivos, permitiendo la programación de las tablas de flujos por parte de las aplicaciones software y especificando cómo migrar el control de la red al controlador.

Un conmutador OpenFlow aprovecha el hecho de que la mayoría de los conmutadores y enrutadores Ethernet actuales contienen tablas de flujo (construidas a partir de memorias TCAM y RAM que realizan la gestión interna) para funciones como cortafuegos, calidad de servicio o recogida de estadísticas. A pesar de que una tabla de flujo es diferente según el fabricante, existe un conjuntos de funciones comunes. El dispositivo OpenFlow dispone de, al menos, tres partes:

- Tabla de flujos. Cada entrada de la tabla dispone de campos en los que buscar coincidencias con los paquetes entrantes, contadores (para estadísticas de los paquetes) e instrucciones sobre qué hacer con los coincidentes.

- Canal seguro. Conecta el dispositivo al controlador, permitiendo el envío de comandos y paquetes entre ambos mediante el protocolo OpenFlow.

- Protocolo OpenFlow. Provee una manera abierta y estándar en la comunicación entre el conmutador y el controlador, permitiendo a este último insertar, eliminar, modificar y buscar en las entradas de la tabla de flujos a través del canal seguro.

3.1 Versiones de OpenFlow

El *OpenFlow Consortium* [27], un grupo colaborativo de investigadores universitarios y administradores de red liderado por miembros de Stanford y Berkeley, lanzó la primera implementación de referencia en noviembre de 2007, con la versión 0.1.0. Se continuó con versiones aún experimentales: 0.2.0, 0.8.0, 0.8.1, 0.8.2, 0.8.9, publicadas en 2008 y 0.9 a mediados de 2009.

En diciembre de 2009 se lanzó la versión 1.0.0, la más ampliamente adoptada, que tiene en cuenta los siguientes campos de cabecera presentes en un paquete Ethernet:

Puerto entrante
Dirección MAC Origen
Dirección MAC Destino
Tipo Ethernet
Id VLAN
Prioridad CoS VLAN
IP Origen
IP Destino
Protocolo IP
Bits ToS (tipo de servicio) IP
Puerto TCP/UDP origen
Puerto TCP/UDP destino

Tabla 2. Campos de cabecera a analizar en la versión 1.0.0 [28]

El proceso de un paquete que entra al conmutador es el siguiente:

- Los campos de cabecera de los paquetes son extraídos y usados para buscar coincidencias.

- Se compara con las reglas definidas para cada entrada en la tabla de flujos OpenFlow, teniendo en cuenta el orden descendente de prioridad. Un paquete puede ser coincidente con una entrada particular de la tabla de flujos usando uno o más campos del paquete. Un campo en la tabla de flujos puede tener el valor "ANY" o cualquiera, que coincidirá con todos los paquetes.

- Si hay una coincidencia, las acciones especificadas en esa entrada se realizan sobre el paquete. Si no, los primeros 200 bytes del paquete son enviados al controlador para su procesado.

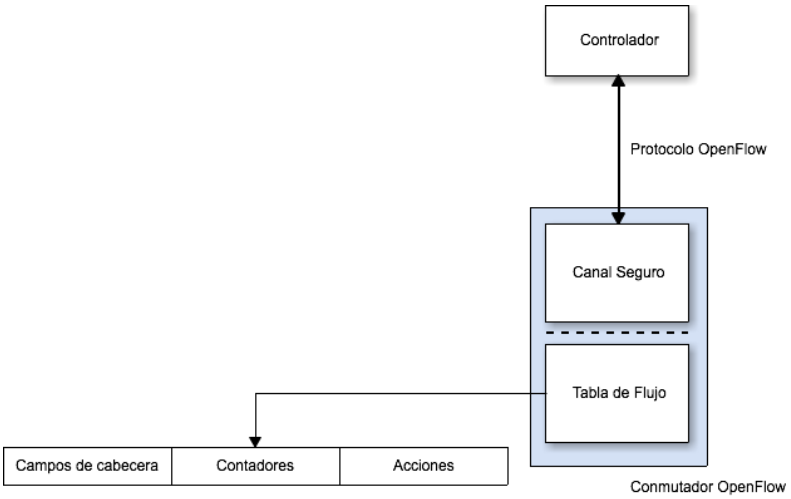

Figura 8. Conmutador OpenFlow y tabla en la especificación 1.0.0 [28]

En la especificación 1.1.0, el conmutador tiene varias tablas de flujos conectadas entre ellas mediante procesamiento en 'pipeline' o tubería y una tabla de grupo. La tabla de grupo está diseñada para realizar operaciones que son comunes entre múltiples flujos, lo que significa que las acciones pertenecientes a un conjunto de ellos son agrupadas.
Los campos en los que se buscan coincidencias también cambian, añadiéndose tres nuevos: metadatos, usado para pasar información entre las tablas cuando un paquete las atraviesa; y etiquetas MPLS y MPLS EXP de clase de tráfico, ambas para dar soporte al protocolo MPLS.
Cuando un paquete entra al dispositivo, se envía y se busca en la primera tabla. Si hay una coincidencia, se trata el paquete y si apunta a otra tabla, se envía hasta ella, proceso que acaba cuando ya no se apunta más. Además, gestiona instrucciones en lugar de acciones. Antes una acción era asociada a cada entrada de la tabla de flujos, que podía ser reenviar el paquete o descartarlo, así como procesarlo normalmente como un conmutador normal. Las instrucciones son más complejas, e incluyen modificar el paquete, actualizar un conjuntos de acciones o actualizar los metadatos.

La especificación 1.2 incluye soporte a direcciones IPv6, así como la posibilidad de conectar un conmutador a múltiples controladores, que pueden comunicarse también entre ellos. Esto provee de recuperación más rápida durante fallos. Asimismo, incorpora balance de carga.

La especificación 1.3.0 incluye la posibilidad de controlar la tasa de paquetes mediante medidores de flujo. Además se añaden conexiones auxiliares entre el conmutador y el controlador. También se incorporan cookies y campos de duración en las estadísticas.

Las especificaciones 1.4.0 y 1.5.0 añade nuevas características como monitorización de flujos, "pipeline" o tubería sensible al tipo de paquetes y estadísticas extensibles en las entradas de las tablas de flujo.

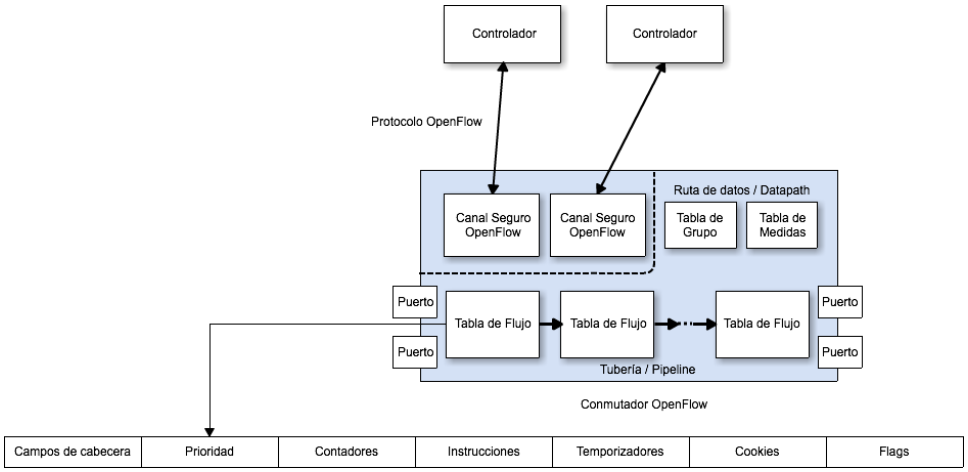

Figura 9. Conmutador OpenFlow en la especificación 1.5.0 [29]

3.2 Implementación y aplicación de OpenFlow

Al ser un concepto nuevo y sin estándares claros, siguen apareciendo nuevas implementaciones constantemente, tanto en el mercado como en el campo de la investigación. Se puede diferenciar según:

- Hardware. Su principal ventaja es el mayor rendimiento respecto a la implementación software. Hay que tener en cuenta que el plano de datos es definido según el fabricante. Una aplicación puede funcionar en un conmutador y no en otro, ya que no todos implementan las mismas características de OpenFlow.

Fabricante	Productos
Arista	Arista EOS, Arista 7124FX
Ciena	Actualización Core Director firmware 6.1.1 en dispositivos Ciena
Cisco	Cisco cat6k, series catalyst 3750, 6500
Juniper	Juniper MX-240, T-640
HP	HP series procurve: 5400 zl, 8200 zl, 6200 yl, 3500 yl, 6600
NEC	NEC IP8800
Pronto	Pronto 3240, 3290
Toroki	Toroki Lightswitch 4810
Dell	Dell Z900 y S4810
Quanta	Quanta LB4G

Tabla 3. Conmutadores hardware OpenFlow [25]

- Software. El coste respecto a los dispositivos hardware es su mejor baza. Los dos proyectos más importantes son: NOX, el primer controlador OpenFlow y Mininet, un entorno virtual que se detallará más adelante. Existen controladores software como ProgrammableFlow Controller de NEC, Big Network Controller de Big Switch o proyectos "open source" como OMNI, Trema, Ryu, FloodLight u OpenDayLight. En cuanto al conmutador, el más aceptado es Open vSwitch.

También, cabe citar algunas características de OpenFlow adicionales a las propias de SDN, como:

- El tráfico experimental y el de producción pueden compartir el mismo conmutador, definiendo por una parte, en una entrada de la tabla de flujo, la acción de enviar el paquete al 'datapath' o procesamiento normal del dispositivo; por otra, el tratamiento de investigación.

- El controlador tiene un conocimiento amplio de la red, y no local como es tradicional. Actualiza dinámicamente el reenvío, ya que puede modificar las entradas de la tabla de flujo en cualquier momento y en caso de fallo de enlace, este elemento puede encontrar nuevas rutas más rápidamente. Mediante herramientas como FlowVisor se puede compartir el control de un conmutador entre varios controladores o balance de carga con ejemplos como Plug-n-Serve.

- Abstracción de flujos. Con una sola entrada en la tabla, se puede gestionar un flujo de tráfico completo.

4. Mininet

4.1 Consideraciones Previas

Mininet [30] es un emulador de red que posee una colección de hosts finales (los cuales ejecutan software Linux estándar), conmutadores, enlaces y controladores en un solo núcleo Linux, utilizado para realizar rápidamente experimentos con SDN y OpenFlow. Mediante una interfaz de línea de comandos (CLI) se puede interactuar y verificar la red.

Ya que Mininet provee de un entorno simulado para la experimentación, se pueden desarrollar y probar nuevas ideas antes de desplegarlas en un entorno real. Además, está abierto a la corrección de fallos por parte de cualquier usuario, se sigue actualizando y se distribuye bajo licencia BSD de código abierto permisivo.

Mininet destaca por:

- Sencillez. Con un solo comando `sudo mn` se puede desplegar una red completa rápidamente.

- Personalización. Se puede crear cualquier tipo de topología. Además, mediante el lenguaje de programación Python, se puede experimentar y almacenar nuevos ejemplos.

- Entorno real. Se emula el envío de paquetes a través de interfaces reales Ethernet, con su velocidad de enlace y retardo. Se permite el diagnóstico con herramientas como ping o tcpdump para comprobar la conectividad entre dos hosts.

4.2 La importancia del controlador en SDN

4.2.1 Red sin controlador y dpctl

Una vez dentro de la máquina virtual de Mininet (ver la instalación en el Anexo), se va a proceder a probar un concepto citado en SDN, que es la separación del plano de control y de datos en un conmutador.
Para ello, hay que empezar mostrando la ayuda de la utilidad mn mediante `sudo mn -h` , apareciendo el mensaje de ayuda:

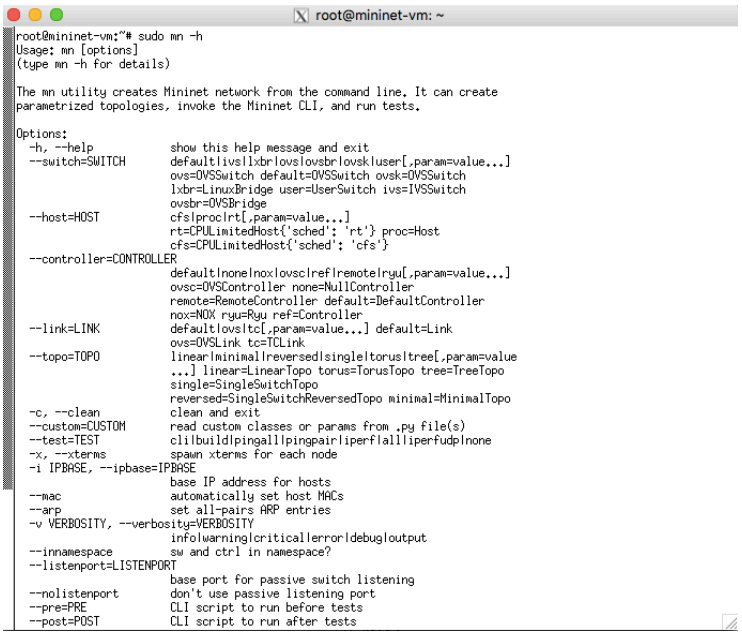

Figura 10. Opciones de ejecución en mininet.

Se pueden destacar las siguientes:

`--switch` = SWITCH. Por defecto trabaja con Open vSwitch y se le puede indicar también con la opción `ovsk`. Si en cambio se escribe `user`, se implementa un conmutador más lento.

`--controller` = CONTROLLER. Por defecto trabaja con un controlador OpenFlow y que está preinstalado en la máquina virtual. Con `nox` y `ryu` se trabaja con los controladores del mismo nombre y con `remote` no se crea un controlador, sino que se mantiene a la escucha de algún controlador remoto.

`--topo` = TOPO. Se crea una topología de red virtual, teniendo diferentes opciones: `minimal` crea una topología por defecto de un conmutador y dos hosts; `single,X` un solo conmutador conectado a X hosts; `linear,X` crea X conmutadores conectados uno detrás de otro, cada uno con un host conectado; y `tree,X` un árbol con fanout X, o puntos hijos por cada nodo.

`--mac` . Asigna direcciones MAC fáciles a los hosts de forma automática.

Se puede crear una red fácil escribiendo directamente `sudo mn`, que crea una red virtual consistente en un controlador “c0”, un conmutador “s1” y dos hosts “h1” y “h2”.
Sin embargo, en este primer caso se intentará trabajar con una red sin controlador, lo que hará que un conmutador no sepa cómo reenviar el tráfico

que le llega y necesitará alguna definición a la hora de manejar los flujos. Para hacerlo de manera sencilla y en un entorno de prueba, se utilizará dpctl, una herramienta de configuración que permite el acceso directo y visible sobre una tabla de flujo sin necesidad de trabajar (por el momento) con el controlador. Éste acceso es permitido en los dispositivos OpenFlow a través del puerto por defecto TCP 6634.

Se procede a probar la red de la figura mediante: `sudo mn --topo=single,4 --mac --controller=remote`

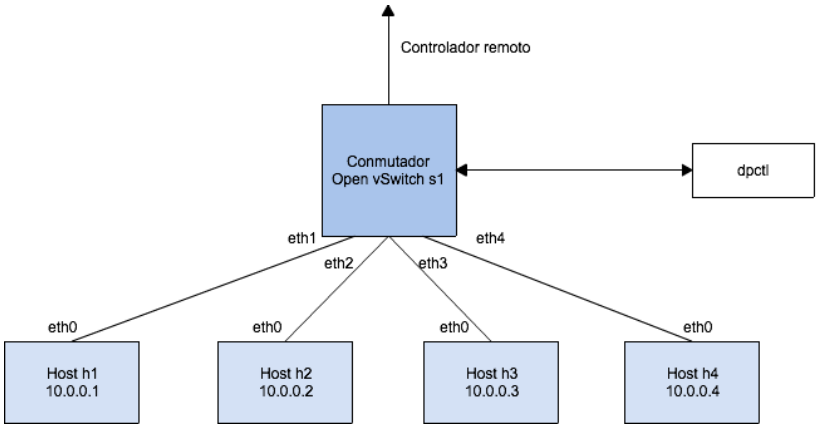

Figura 11. Red virtual creada.

Una vez ejecutada, y después de ver como se crea la red, aparece el *prompt* de mininet, que permite trabajar con la red virtual mediante diferentes nuevas opciones como:

- `help`. Muestra los comandos disponibles en el prompt de mininet.
- `nodes`. Da una lista de los dispositivos virtuales.
- `net`. Devuelve una lista con todos los enlaces.
- `dump`. Información de direcciones o identificadores de la red.
- `xterm`. Si se acompaña del nombre de un host, abre otra ventana para trabajar con el host indicado más cómodamente.

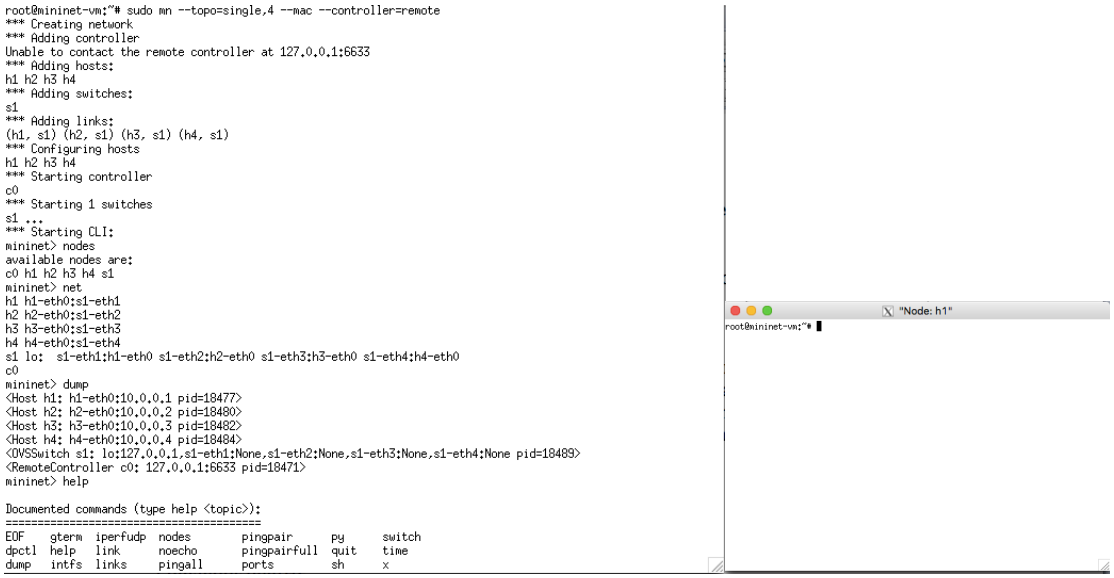

Figura 12. Creación de red y de los comandos nodes, net, dump, help y xterm h1.

Se abre cada uno de los nodos en ventanas diferentes con xterm, y se procede a probar un ping de h1 a h3, y a ejecutar la herramienta tcpdump en éste último para monitorizar los paquetes. El resultado es que todas las pruebas de ping fallan.

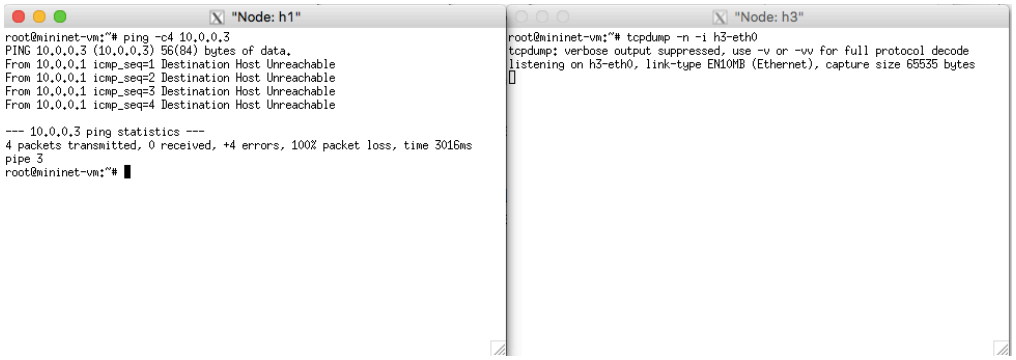

Figura 13. Pruebas de ping entre h1 y h3.

Si se ejecuta la herramienta dpctl (en una nueva terminal conectada a la máquina virtual mininet mediante SSH), se puede ver que la tabla de flujo del conmutador está vacía ya que al no estar conectado a un controlador, no sabe cómo manejar los paquetes que le llegan.

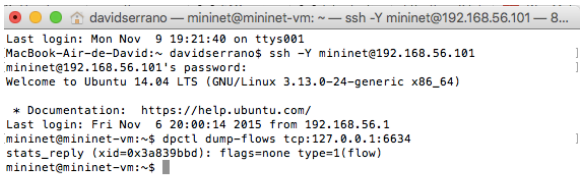

Figura 14. Comprobación de la tabla de flujo en el conmutado mediante dpctl.

Por tanto, en esta nueva ventana dpctl hay que añadir dos flujos (comunicación ping bidireccional) mediante:

```
dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions:output=3
dpctl add-flow tcp:127.0.0.1:6634 in_port=3,actions:output=1
```

Si se realiza el ping entre h1 y h3 de nuevo, se puede comprobar que funciona correctamente. En el resto de hosts se sigue sin recibir nada.

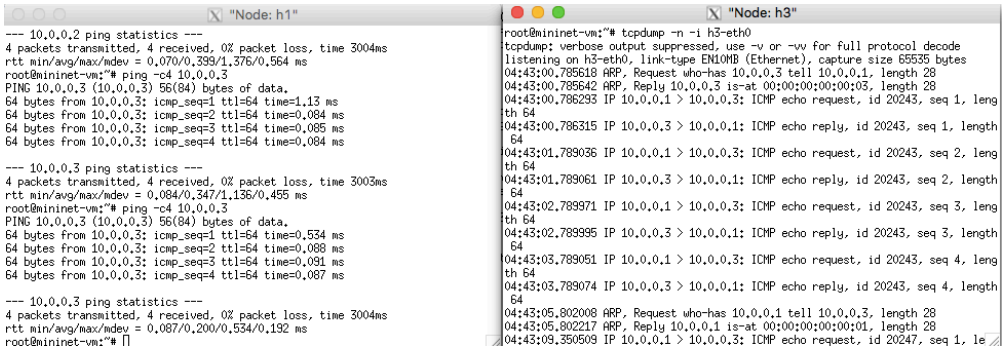

Figura 15. Pruebas de ping entre h1 y h3 tras modificar la tabla de flujo con dpctl.

En [31] se puede ver una lista completa para modificar los flujos de forma manual con dpctl, existiendo una gran variedad de campos a comprobar y acciones.

4.2.2 Red con controlador y mensajes OpenFlow

En este apartado, se utiliza el analizador software Wireshark, se configura para la interfaz de loopback (lo) y se aplica el filtro del protocolo "openflow_v1" que trabaja con OpenFlow en la especificación 1.0.0.

A continuación y con Wireshark capturando, se introduce en la red anterior un controlador OpenFlow mediante `controller ptcp:` que viene por defecto en

Mininet y que actúa como un conmutador que aprende, sin ninguna tabla de flujo instalada previamente:

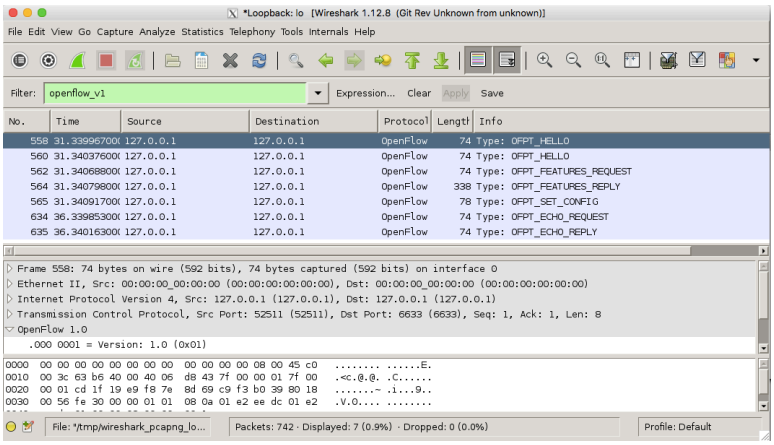

Figura 16. Mensajes OpenFlow al iniciar el controlador en la red.

Todos los mensajes tienen una cabecera común que se indica en la especificación 1.0.0 [28] y que se puede encontrar en la pestaña de "OpenFlow 1.0":

- Version. En este caso, todos los mensajes son de la versión 1.0.

- Type. Tipo de paquete e identificado con un número que es igual para todos los paquetes del mismo tipo. Por ejemplo el HELLO se identifica con el 0, y el SET_CONFIG con 9.

- Length. Longitud del paquete incluyendo la cabecera.

- Transaction ID. Identificación de la transacción asociada al paquete. Las parejas Request – Reply tienen el mismo identificador.

Se pueden apreciar diferentes tipos de mensajes:

- HELLO. Se producen desde el controlador al conmutador y viceversa. Siguiendo el intercambio TCP, el controlador envía su número de versión al dispositivo. Al tener un solo conmutador, podemos saber cuál es el origen y el destino mirando la pestaña del protocolo TCP, y mirando los puertos, ya que el controlador está en el puerto 6633, mientras que el conmutador se encuentra en otro (en este caso, el 52511).

- FEATURES_REQUEST. El controlador pregunta al controlador por qué puertos están disponibles.

- SET_CONFIG. El controlador pide al conmutador que le envíe las expiraciones temporales de los flujos.

- FEATURES_REPLY. El conmutador responde con una lista de puertos, velocidades de puertos, tablas soportadas y acciones.

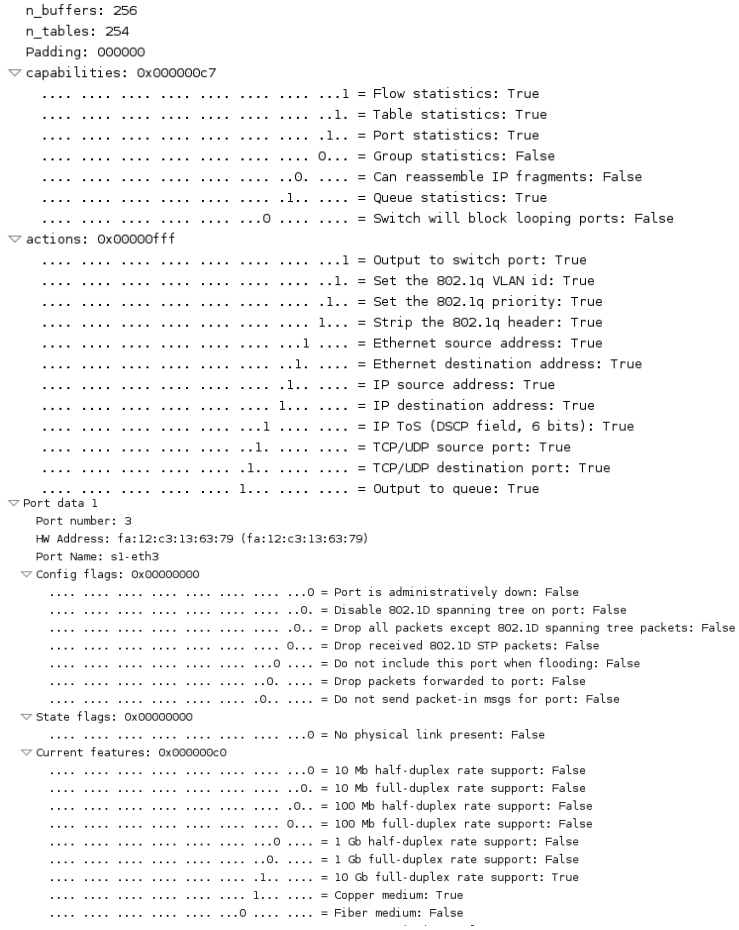

Figura 17. Detalle de tablas y acciones soportadas, así como características de un puerto.

- ECHO_REQUEST y ECHO_REPLAY. Mensajes para mantener la comunicación entre el conmutador y el controlador.

A continuación, se volverá a probar un ping entre h1 y h3, para comprobar diversos aspectos teóricos explicados anteriormente.

En Wireshark, se capturan nuevos tipos de paquetes, como se puede apreciar en la figura 18:

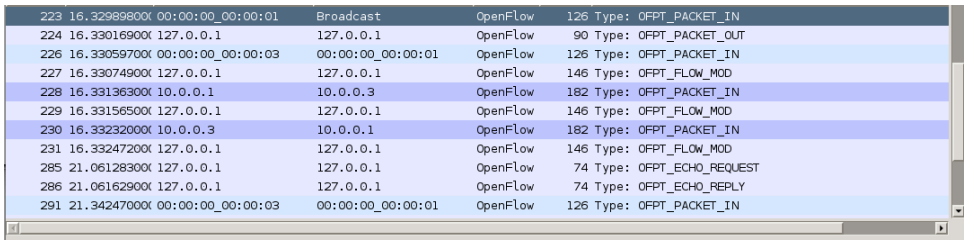

Figura 18. Intercambio de paquetes al realizar un ping entre h1 y h3

- PACKET_IN: Se envía al recibir un paquete que no coincide con ninguna entrada en la tabla de flujo del conmutador, haciendo que se envíe al controlador. Se genera por primera vez un paquete *broadcast* con la petición ARP. Si se analiza la pestaña del protocolo OpenFlow se puede ver la razón de envío del paquete.

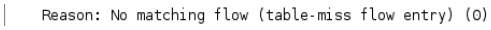

Figura 19. Razón del envío del primer paquete PACKET_IN

Los sucesivos paquetes PACKET_IN no pasarán por el controlador, sino que serán paquetes ICMP normales y el controlador habrá de indicar dos flujos para comunicarse entre h1 y h3.

- PACKET_OUT: El controlador envía un paquete de respuesta al conmutador con ARP con las direcciones MAC.

- FLOW_MOD: El controlador además indica al dispositivo que añada (o modifique) una entrada de la tabla de flujo.

Figura 20. Detalle de un paquete FLOW_MOD

La conclusión de esta prueba es comprobar otra característica teórica de SDN y OpenFlow, y es que cuando al dispositivo llega un paquete que no coincide con ninguna entrada en la tabla de flujo, lo envía al controlador para que le indique qué hacer con él. Una vez que lo sabe, ya no necesita esta comunicación, y realiza el reenvío por su cuenta. Si se realiza de nuevo un ping entre los dos

hosts, se puede apreciar la diferencia de tiempo entre la primera vez y sucesivos intentos.

```
root@mininet-vm:~# ping -c1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=3.11 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.117/3.117/3.117/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.514 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.514/0.514/0.514/0.000 ms
root@mininet-vm:~#
```

Figura 21. Diferencia en el tiempo de respuesta de ping.

4.3 MiniEdit

Como se ha podido observar, la creación de una red virtual en Mininet se realiza mediante la interfaz de línea de comandos. En un entorno más educativo, puede surgir la necesidad de crear la misma red de forma más sencilla, como por ejemplo, una interfaz gráfica.
Como solución, en el propio emulador Mininet, se incluye una carpeta con diversos ejemplos, entre los que destaca MiniEdit, un editor gráfico que hace uso del potencial de Mininet.

Para ejecutarlo, habrá que cerrar todos los procesos creados en la prueba anterior mediante `exit` (salir de la red creada por Mininet), `killall controller` (terminar el proceso del controlador) y `sudo mn -c` (para 'limpiar' cualquier elemento creado en la red anterior). Ahora, desde una terminal conectada mediante SSH a la máquina virtual se ejecuta
`sudo ~/mininet/examples/miniedit.py`

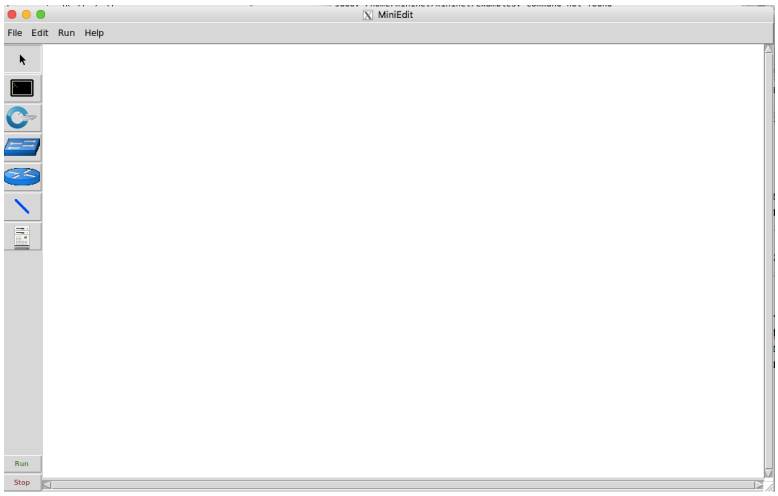

Figura 22. Editor MiniEdit.

Desde la sencilla interfaz que se presenta, se puede observar una serie elementos para construir una red en la parte derecha de la ventana, teniendo los siguientes por orden de aparición:

- Flecha de selección. Se utiliza para mover elementos seleccionados en la red.

- Host. Se crea un nodo que realiza la función de un host como los anteriormente utilizados. Una vez seleccionado este elemento, simplemente haciendo clic en la parte de la derecha se añaden nuevos hosts. Si se pulsa botón derecho y se mantiene se pueden acceder a las propiedades de cada host.

- Conmutador. Se crea un dispositivo OpenFlow Open vSwitch por defecto.

- Conmutador tradicional. Se crea un conmutador Ethernet que aprende con características por defecto y que opera de forma independiente. Tiene desactivado el protocolo de Spanning Tree (peligro con bucles).

- Enrutador tradicional. Se crea un enrutador básico que funciona como un host con el reenvío IP activado. No puede ser configurado desde la interfaz gráfica.

- Enlace de red. Se crea un enlace, pinchando en un elemento y arrastrando hasta el otro. Con el botón derecho se puede configurar las Propiedades del enlace.

- Controlador. Se pueden añadir muchos tipos de controlador, aunque por defecto se crea el controlador de referencia en OpenFlow, que implementa el comportamiento de un conmutador 'learning' de capa 2. En el menú de Propiedades se puede configurar las propiedades del controlador.

- Ejecutar / Parar. Botones para empezar o terminar la simulación. Cuando está en ejecución, aparecen otras propiedades en cada elemento al clicar con el botón derecho.

Se procede a crear una red algo más compleja que la anterior, debido a la sencillez que conlleva esta interfaz gráfica en comparación con los comandos. Además se cambia el número de puerto del controlador c1.

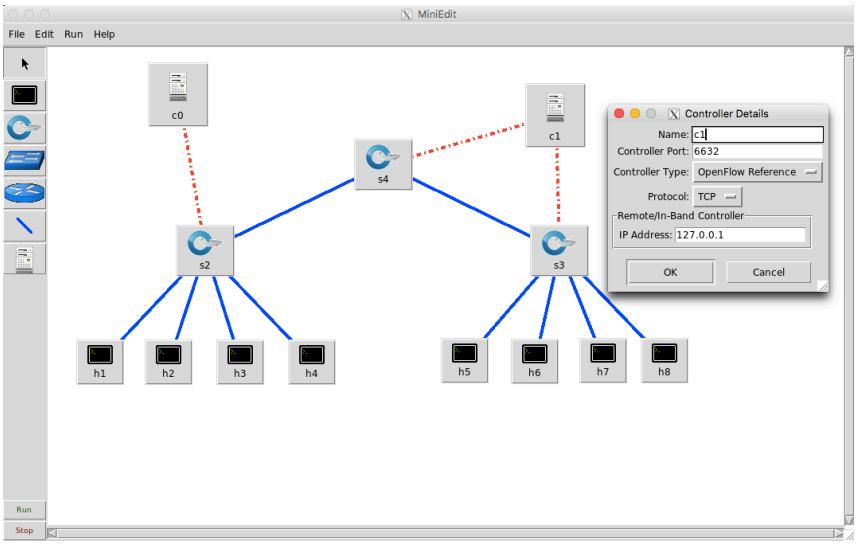

Figura 23. Red creada y propiedades del controlador c1.

Es importante destacar que para tener otras opciones de configuración más específicas, es interesante combinar la interfaz gráfica con la CLI. Esto se realiza activando la opción en *Edit -> Preferences* pudiendo modificar algunas otras opciones como el grupo de IPs con las que trabajar, cambiar el tipo de conmutador OpenFlow, la versión del protocolo o configurar un puerto para la herramienta dpctl.

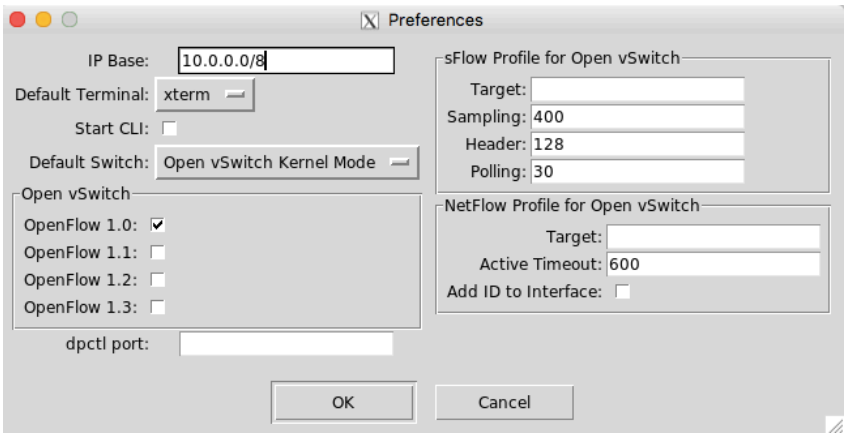

Figura 24. Preferencias de la simulación.

Desde la Terminal con la que se ha lanzado MiniEdit y al haber activado la interfaz en línea de comandos, se verá cómo se configura la red, para posteriormente volver a tener el prompt 'mininet>'. Por tanto, se tiene la misma interfaz que antes, con el consecuente ahorro de tiempo al hacerlo de forma gráfica.

```
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
<class 'mininet.node.Host'>
Getting Links.
*** Configuring hosts
h3 h7 h6 h1 h8 h2 h4 h5
**** Starting 2 controllers
c0 c1
**** Starting 3 switches
s2 s3 s4
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet>
mininet>
```

Figura 25. Visualización de la topología en MiniEdit con CLI.

Mediante la opción *Run -> Show OVS Summary* se puede comprobar la configuración de puertos y enlaces en todos los disposivos de la red.

Figura 26. Comprobación mediante OVS Summary.

Otra característica interesante es que mediante *Run -> Root Terminal* se abre una nueva ventana xterm conectada a la máquina virtual Mininet de la red. Desde aquí se comprobará las tablas de flujo de los conmutadores, observando las tablas vacías.

```
root@mininet-vm:~/mininet/examples# sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~/mininet/examples# sudo ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~/mininet/examples# sudo ovs-ofctl dump-flows s4
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~/mininet/examples#
```

Figura 27. Tablas de flujo de los conmutadores en la red virtual.

Como en el apartado anterior, se probará un ping para crear entradas en las tablas de los conmutadores OpenFlow. Se ejecutará Wireshark en el host h1, para monitorizar los paquetes intercambiados.
En la ventana xterm de h1 se lanza `ping -c3 10.0.0.8` , se observa los paquetes intercambiados, y se mira las tablas de los conmutadores de nuevo, observando las entradas que se han añadido a todos los conmutadores involucrados.

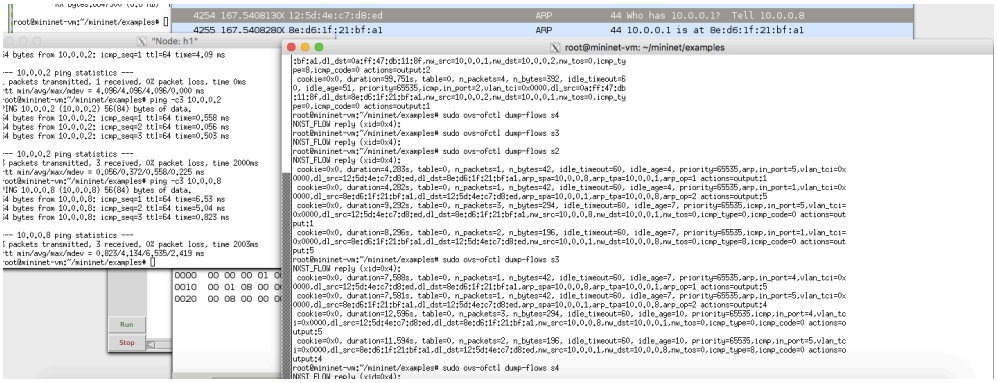

Figura 28. Ping con origen en h1, monitorización con Wireshark y tablas de flujo.

Para finalizar la ejecución, como se indica en el aviso de la ventana CLI, es escribir `exit` en el prompt, y después pulsar en Stop en MiniEdit.

Finalmente, es interesante guardar topologías y configuraciones para estudiarlas posteriormente. Se puede hacer de dos formas:

- Como *.mn, para ejecutarlo cuando se desee en MiniEdit. Se realiza con *File -> Save* y se ejecuta con *File -> Open.*

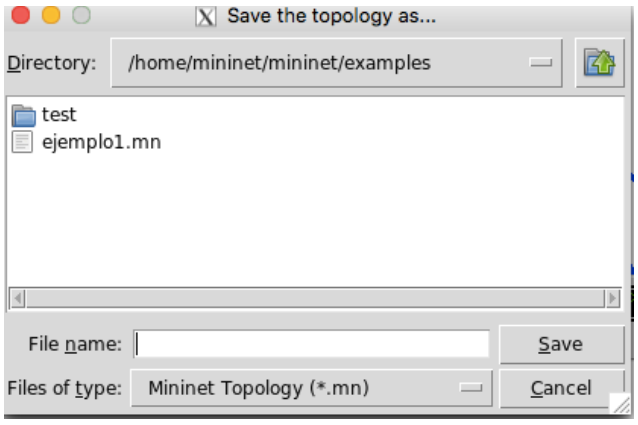

Figura 29. Guardar el fichero *.mn

- Como *.py, para ejecutarlo cuando se desee desde la terminal CLI. Se hace con *File -> Export Level 2 Script.*

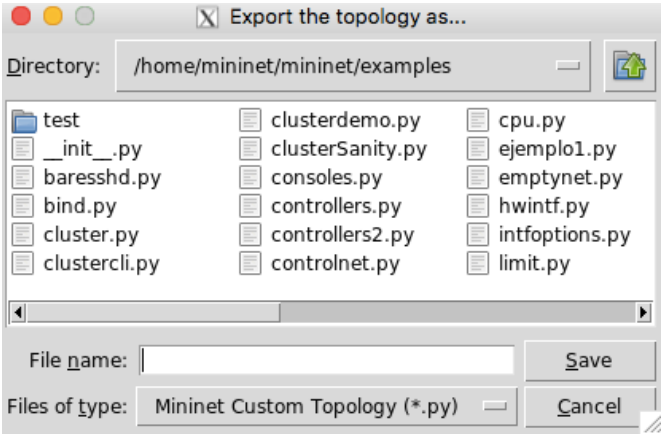

Figura 30. Guardar el fichero *.py

En este segundo caso, se debe volver a abrir la topología mediante CLI con:

`sudo chmod 777 ejemplo1.py` (lo hace ejecutable)
`sudo ./ejemplo1.py`

```
root@mininet-vm:~/mininet/examples# sudo ./ejemplo1.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h3 h7 h6 h1 h8 h2 h4 h5
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> nodes
available nodes are:
c0 c1 h1 h2 h3 h4 h5 h6 h7 h8 s2 s3 s4
mininet>
```

Figura 31. Ejecución de la red virtual guardada.

4.4 Controlador POX como conmutador 'que aprende'

4.4.1 Definiciones

Para finalizar, se pretende realizar un sencillo controlador que actúe como un conmutador 'learning' mediante la plataforma POX.

Como se sabe, un conmutador de capa 2 aprende las direcciones MAC (o Ethernet) de los nodos de la red escuchando las direcciones de los paquetes que pasan por él.
A modo de ejemplo, se piensa en dos hosts A y B conectados mediante un conmutador a dos puertos P1 y P2 y el cual no sabe las direcciones de los hosts. A quiere comunicarse con B y mediante ARP pregunta la MAC de la dirección IP que conoce, luego envía la trama al conmutador, que hace un *broadcast* por

todos sus puertos excepto por el que le llegó, para saber la MAC de B y a través de qué puerto llega. También almacena que a través de P1 se puede alcanzar el host A.
En el *broadcast*, B recibe la trama, la procesa y contesta a A con un paquete que recibe el conmutador, el cual guarda que a través de P2 llega al nodo B, y envía el paquete directamente a P1, ya que sabe que por ese puerto llega a A.

POX es la versión en lenguaje Python de NOX, la primera plataforma pensada para construir aplicaciones que controlen la red en OpenFlow. Está creada para el desarrollo rápido y control de software de la red, a nivel básico y con un rendimiento similar a NOX.

4.4.2 Creando un conmutador de capa 2

- En primer lugar, habrá que eliminar cualquier controlador y pruebas realizadas con anterioridad y crear una nueva red de prueba, por tanto desde la ventana xterm se ejecuta `killall controller  sudo mn -c` y `sudo mn --topo=single,4 --mac --switch=ovsk --controller=remote`

- A continuación, hay que descargar el repositorio de POX y ejecutar el código base incluido desde la terminal, mediante:
  `git clone http://github.com/noxrepo/pox`
  `cd pox`
  `./pox.py log.level --DEBUG misc.of_tutorial` (muestra los logs en la terminal y ejecuta el of_tutorial.py incluido en la carpeta de ejemplos de NOX "misc")

Figura 32. Ejecución de of_tutorial.py

Este primer ejemplo actúa como un *hub*, es decir, como un concentrador que reenvía los paquetes por todos sus puertos excepto por el que le llegó.

El comportamiento de este dispositivo se comprueba mediante un ping desde h1 a h4 y tcpdump en el resto de hosts. Como se puede ver en la figura, todos los hosts reciben los paquetes ping y ARP de la comunicación entre los dos hosts.

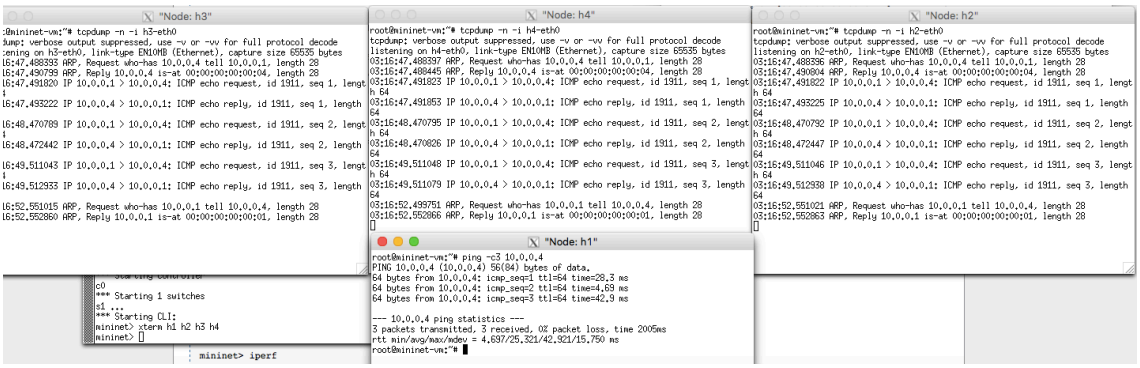

Figura 33. Ping entre h1 y h4 con el controlador actuando como un hub.

- Se procede a finalizar la actuación del controlador y la ejecución de la red virtual, para poder analizar el código de of_tutorial.py mediante:
`sudo vi pox/pox/misc/of_tutorial.py`

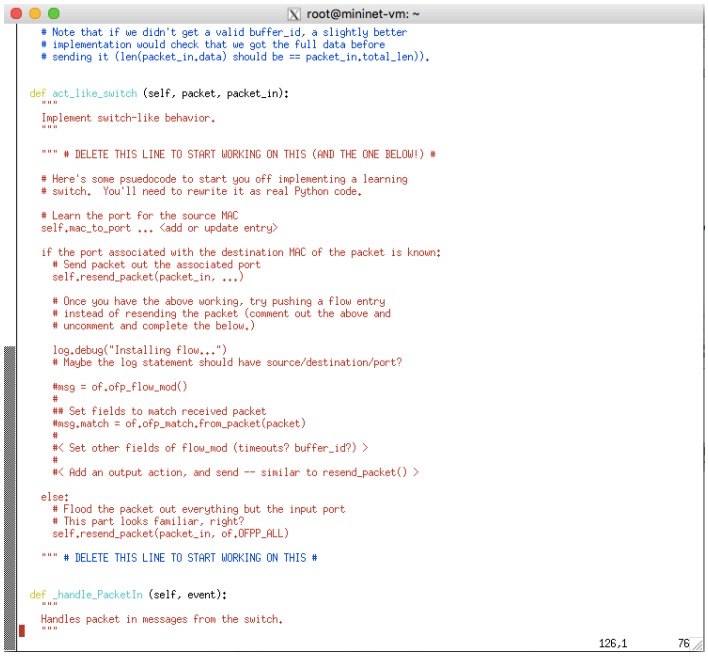

Figura 34. Detalle del comportamiento del conmutador.

Como se puede apreciar en los comentarios, el proceso es guiado y conlleva modificar algunas partes del código.

- El funcionamiento es sencillo de describir: se extrae la dirección MAC del paquete entrante y se almacena en la tabla MAC – puerto. Se extrae la dirección MAC destino; mediante un bucle if – else se comprueba si está ya en la tabla, caso en el que se almacena, se muestra un mensaje 'log' y

se actualiza en la tabla de flujo. Si no está en la tabla, se realiza el mismo comportamiento que el hub o concentrador probado antes.

- Con la ayuda del tutorial OpenFlow para manejar el ejemplo del controlador POX [32] se ha reescrito la parte del conmutador y se ha comprobado el funcionamiento del mismo con ping entre los hosts.

```
def act_like_switch (self, packet, packet_in):

  self.mac_to_port[str(packet.src)]=packet_in.in_port # Almacena la MAC del puerto origen

  if str(packet.dst) in self.mac_to_port: #Compara la MAC destino con la tabla, si esta entra en el if

    port = self.mac_to_port[str(packet.dst)] # Almacena en port la MAC destino (es un integer)

    log.debug("Creando flujo  %s.%i > %s.%i"% (packet.src, packet_in.in_port, packet.dst,port))
    # Loging para mostrar como dos string y dos integer

    fm = of.ofp_flow_mod() #Se crea un objeto ofp_flow_mod para crear un flujo
    fm.match.dl_dst = packet.dst
    fm.match.dl_src = packet.src # match es un objeto ofp_match en el que se indica los puertos
    fm.idle_timeout = 20 # Tiempo de inactividad
    fm.hard_timeout = 40 # Tiempo en el que se elimina definitivamente
    fm.actions.append(of.ofp_action_output(port=port)) # La unica accion que realiza es enviar por el puerto destino
    fm.buffer_id = packet_in.buffer_id # Ayuda en el tutorial OpenFlow
    self.connection.send(fm) # Envia el mensaje OpenFlow al conmutador

  else:
    self.resend_packet(packet_in, of.OFPP_ALL) # Si no esta en la tabla, realiza el comportamiento de un HUB
```

Figura 35. Funcionamiento del controlador POX como un conmutador de capa 2.

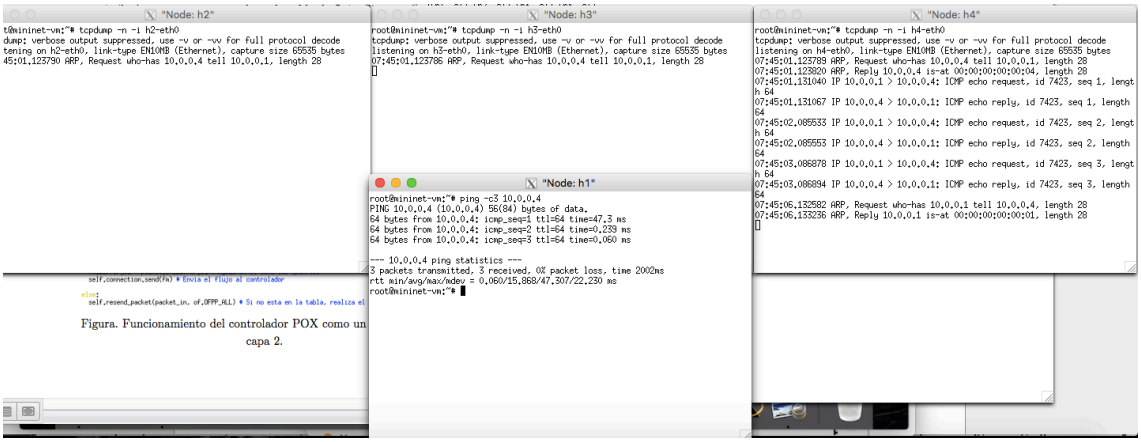

Figura 36. Ping entre h1 y h4.

```
DEBUG:misc.of_tutorial:Creando flujo  00:00:00:00:00:01.1 > 00:00:00:00:00:04.4
DEBUG:misc.of_tutorial:Creando flujo  00:00:00:00:00:04.4 > 00:00:00:00:00:01.1
DEBUG:misc.of_tutorial:Creando flujo  00:00:00:00:00:02.2 > 00:00:00:00:00:01.1
DEBUG:misc.of_tutorial:Creando flujo  00:00:00:00:00:01.1 > 00:00:00:00:00:02.2
```

Figura 37. Loging de la creación de flujos.

5. Conclusiones y líneas de trabajo futuras

A lo largo del presente trabajo se ha intentado mostrar el contexto de aparición de las Redes Definidas por Software, un concepto innovador en las redes actuales y que procede de años de pruebas y de investigación.
Se ha visto como, a pesar de que haya ideas innovadoras o con notables beneficios teóricos, su implementación práctica y a gran escala es la verdadera prueba, teniendo mucha importancia la adopción por parte de los fabricantes, que son reticentes ante innovaciones muy drásticas que pongan en compromiso su mercado y estabilidad o que no les dé confianza al ser experimentadas en pequeños laboratorios. Aún así, parece que poco a poco la comunidad va aceptando los beneficios evidentes de SDN, teniendo ejemplos como las más de 95 empresas miembro actuales en la *Open Networking Foundation*, o B4, una WAN que conecta los *data centers* de Google y basada en OpenFlow.

Se han descrito las dos principales características de SDN: la programabilidad, con pruebas anteriores como las Redes Activas, y los intentos de separar el plano de control y de datos en el enrutamiento. Además, se han citado algunas características y aplicaciones reales, para mostrar sus beneficios.

A continuación, se ha descrito su implementación *de facto*, OpenFlow, con las sucesivas versiones, ejemplos de productos reales tanto en hardware como en software, así como modo básico de funcionamiento.

Cabe destacar que se trata de un protocolo muy joven (como se ha comentado, su primera versión es de finales de 2007) y por tanto aún se está desarrollando; no hay elementos dominantes en su arquitectura, no hay un software o hardware especialmente recomendado y en cuanto a sus características, aún tiene desafíos como:

- Controlador. Al ser un único elemento centralizado, comprometer su seguridad es un aspecto crítico, ya que se tiene el conocimiento completo de la red. En las redes tradicionales un dispositivo se autogestiona y decide teniendo una visión local de la red, mientras que en SDN se necesita tener disponible la comunicación con el controlador (lo que puede significar también que se convierta en un punto clave de escalabilidad, un cuello de botella). Además, el tener réplicas o más controladores hace necesaria una comunicación coordinada entre ellos, ya que puede haber inconsistencias en las tablas de flujo.

- Conmutador. Como se ha comentado, la implementación del plano de datos en el conmutador no está especificada en OpenFlow, por lo que una aplicación puede funcionar en un dispositivo de un fabricante, y no en el de otro. Por tanto, la compatibilidad de dispositivos es un aspecto importante.

Finalmente se ha probado el software de emulación Mininet, una herramienta orientada a demostrar de manera sencilla y educativa las principales características de OpenFlow y el modo en el que interactúan sus diferentes elementos de red, sobretodo el controlador.

SDN y en concreto OpenFlow se está comenzando a desplegar y adoptar en el momento justo debido principalmente a que la situación actual en las redes requería de algún cambio, o de al menos un intento de ello. Las posibilidades y beneficios de este nuevo concepto son muy grandes; sin embargo, queda mucho trabajo por hacer y definir. Si en algo más de cinco años se han concebido tantas propuestas e implementaciones dentro de este campo, es interesante pensar lo que les depara a las redes de datos en el futuro más próximo.

6. Bibliografía

[1] López, Diego R. “Internet. La Red con mayúsculas” (1997): 13-14.

[2] Feamster, N.; Rexford, J. y Zegura, E. (2013) “The Road to SDN: An Intellectual History of Programmable Networks” https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf (Consulta: 3 julio de 2015).

[3] Larrabeiti, D.; Calderón, M.; Azcorra, A. y García A. (2011) “Redes Activas con IPv6” http://www.it.uc3m.es/maria/papers/redact_com_world01.pdf (Consulta: 10 de julio de 2015).

[4] Sterbenz, James P.G. et al. (2010) “The Great Plains Environment for Network Innovation (GpENI): A Programmable Testbed for Future Internet Architecture Research” http://www.lancs.ac.uk/~alia3/GpENI.pdf (Consulta: 10 de julio de 2015).

[5] Calderón, M.; Sedano, M.; Eibe, S. (2007) “Principios y Aplicaciones de las Redes Activas” http://www.it.uc3m.es/maria/papers/apli_RA_jitel99.pdf (Consulta: 12 de julio de 2015).

[6] Alexander, S. “A Generalized Computing model of Active Networks” (1998).

[7] Calvert, K. “Reflections on Network Architecture: an Active Networking Perspective” (2006) http://ccr.sigcomm.org/archive/2006/april/p27-calvert.pdf (Consulta: 12 de julio de 2015).

[8] Yasuda, H. “Active Networks: Second International Working Conference, IWAN 2000” (2000): 2.

[9] Colaboradores de CT3. Control and Data plane [en línea]. Communications Technologies Tips and Tricks (CT3). http://wiki.nil.com/Control_and_Data_plane (Consulta: 15 de julio de 2015).

[10] Lakshman, T.V.; Nandagopal T.; Ramjee R.; Sabnani K.; Woo, T. “The SoftRouter Architecture” http://conferences.sigcomm.org/hotnets/2004/HotNets-III%20Proceedings/lakshman.pdf (Consulta: 18 de julio de 2015).

[11] Nait Takourout R.; Pierre, S; Marchand, L. (2006) “Separation of the Control Plane and Forwarding Plane in Next-Generation Routers” http://www.larim.polymtl.ca/pdf/52.pdf (Consulta: 20 de julio de 2015).

[12] Biswas J. et al. (1998) “The IEEE P1520 Standards Initiative for Programmable Network Interfaces” http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=722138 (Consulta: 20 de julio de 2015).

[13] Yan, H.; Maltz, D.; Eugene, Ng T.S.; Gogineri, H.; Zhang, H.; Cai, Z (2007) “Tesseract: A 4D Network Control Plane” http://www.cs.cmu.edu/~4D/papers/tesseract-nsdi07.pdf (Consulta: 20 de julio de 2015).

[14] Colaboradores de Wikipedia. “Network Processor” (en línea). Wikipedia, the free encyclopedia https://en.wikipedia.org/wiki/Network_processor (Consulta: 20 de julio de 2015).

[15] Casado, M. et al. "SANE: A Protection Architecture for Enterprise Networks" (2006): 137-151.

[16] Casado, M. et al. "Ethane: Taking control of the enterprise" (2007): 1-12.

[17] Cisco, “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper”. (2015) http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html (Consulta: 3 de agosto de 2015).

[18] Zander, J.; Forchheimer, R. “The SOFTNET Project: a retrospect” (1988): 343-345.

[19] Colaboradores de Wikipedia. “Computación en la nube” (en línea). Wikipedia, la enciclopedia libre https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube (Consulta: 5 de agosto de 2015).

[20] HP, “HP OpenFlow and SDN technical overview” (2013) http://h17007.www1.hp.com/docs/networking/solutions/sdn/devcenter/02_-_HP_OpenFlow_and_SDN_Technical_Overview_TSG_v1_2013-10-01.pdf (Consulta: 5 de agosto de 2015).

[21] Juniper Networks, “What’s behind network downtime?” (2008) https://www-935.ibm.com/services/au/gts/pdf/200249.pdf (Consulta: 6 de agosto de 2015).

[22] Cisco, “Cisco Visual Networking Index: Forecast and Methodology, 2014-2019” (2015) http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf (Consulta: 7 de agosto de 2015).

[23] ONF, “What is ONF” (2013) https://www.opennetworking.org/images/stories/downloads/about/onf-what-why.pdf (Consulta: 10 de agosto de 2015).

[24] Meru Networks, “Demystifying Software-Defined Networking for Enterprise Networks” (2013) http://www.merunetworks.com/collateral/solution-briefs/an-introduction-to-sdn-sb.pdf (Consulta: 10 de agosto de 2015).

[25] Xia, W.; Weng, Y.; Foh, C. H.; Niyato, D.; Xie, H. (2015) “A Survey on Software-Defined Networking” http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6834762 (Consulta: 11 de agosto de 2015)

[26] Jeong, K.; Kim, J.; Kim, Y; “QoS-aware network operating system for software defined networking with generalized OpenFlows” (2012). 1167-1174.

[27] McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. (2008) “OpenFlow: Enabling Innovation in Campus Networks” http://archive.openflow.org/documents/openflow-wp-latest.pdf (Consulta: 20 de agosto de 2015).

[28] ONF, “OpenFlow Switch Specification, Version 1.0.0” (2009) https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf (Consulta: 15 de septiembre de 2015).

[29] ONF, “OpenFlow Switch Specification, Version 1.5.0” (2014) https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf (Consulta: 16 de septiembre de 2015).

[30] Lantz, B.; Handigol, N.; Heller, B.; Jeyakumar, V.; (2015) “Introduction to Mininet” https://github.com/mininet/mininet/wiki/Introduction-to-Mininet (Consulta: 20 de septiembre de 2015).

[31] Colaboradores de GitHub. “Dpctl Flow Mod Cases” (en línea) https://github.com/CPqD/ofsoftswitch13/wiki/Dpctl-Flow-Mod-Cases (Consulta: 7 de noviembre de 2015).

[32] Open Networking Foundation. “OpenFlow Tutorial” (2011) http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial (Consulta: 12 de noviembre de 2015).